A Generic Persistence Model for (C)LP Systems

J. Correas*, J. M. Gómez*, M. Carro*, D. Cabeza*, and M. Hermenegildo*,**

(*) School of Computer Science, Technical University of Madrid (UPM)
(**) Depts. of Comp. Science and El. and Comp. Eng., U. of New Mexico (UNM)

Mutable state is traditionally implemented in Prolog and other (C)LP systems by performing dynamic modifications to predicate definitions at runtime, i.e. to *dynamic predicates* of the *internal database*. Dynamic facts are often used to store information accessible per module or globally and which can be preserved through backtracking. These database updates, despite the obvious drawback of their non-declarative nature, have practical applications and they are given a practical semantics by the so-called logical view of (internal) database updates.

On the other hand, the lifetime of the data in the Prolog internal database is that of the Prolog process, i.e., the Prolog database lacks *persistence*. In this context persistence means that program state modifications will survive across program executions, and may even be accessible to other programs—atomically and concurrently. Traditionally, this has been taken care of explicitly by the programmer by, e.g., periodically reading and writing state to an external device (a file or an external database through a suitable interface) and providing locking for concurrency. This approach offers a workable but very tedious solution, where significant modifications to programs are needed and where, unless substantial effort is invested, only limited functionality is achieved.

The fundamental idea that we propose is to make persistence be *a characteristic of certain dynamic predicates*, which encapsulate the persistent state, and to automate implementation by coding persistence once and for all in a reusable (system) library providing the class of *persistent predicates*. The main effect of *declaring a predicate persistent* (a process for which we propose a suitable syntax, compatible with the Ciao system's assertion language) is that *any changes made to such predicates persist from one execution to the next one, and are transactional, and, optionally, externally visible*. The model allows associating an external, persistent storage medium (a file, a database table, etc.) to each such predicate, which will "reside" in that medium. Notably, persistent predicates appear to a program as ordinary (dynamic) predicates: calls to them do not need to be coded or marked specially, and the builtins to update them are (suitably modified versions of) the same used with the internal database (e.g., `asserta/1`, `assertz/1`, `retract/1`, etc.). Thus, only minor modifications to the program code (often independent of its internal logic) are needed to achieve persistence. Also, when using persistent predicates the external storage is at all times *in sync* with the internal database. This provides security against, e.g., system crashes or abnormal termination. Also, transaction atomicity allows *concurrent access* to be handled with only limited effort. Thus, files and/or external databases can be used to communicate and share data among programs, which

each view as part of their internal databases. Quite interestingly, since persistent predicates are viewed as regular dynamic Prolog predicates, analyzers (and related tools) can deal with them with no additional effort. In turn, information deduced by analysis tools (such as, e.g., types and modes) can be used to *optimize* accesses to external storage (the full paper provides performance data for such optimizations in the context of a relational, SQL database).

Finally, perhaps the most interesting advantage of the notion of persistent predicates is that it *abstracts away* the storage mechanism. This allows developing applications which can store data alternatively on, e.g., files or databases with only a few simple changes to a declaration stating the location and modality used for persistent storage. It also minimizes impact on the host language, as the semantics of the access to the database is *compatible* with that of Prolog. We also argue that the conceptual model of persistence developed provides one of the most natural way of interfacing logic programs with databases.

A number of current Prolog systems offer features which are related to the capabilities offered by our approach: Quintus Prolog has *ProDBI* (also available for SICStus under the generic name *Prodata*), which allows queries (but not updates) on tables in a similar way to Prolog predicates. SICStus Prolog has also special interfaces to database systems. XSB and SWI include *PrologSQL*, which can compile on demand a conjunction of literals to SQL using the compiler by Draxler, also used in our approach, but which do not provide transparent persistence. However, we argue that none of these approaches achieve the same level of flexibility, conceptual simplicity, and seamless integration with Prolog achieved by our proposal.

Implementations of our proposed model have been used in several non-trivial applications, such as the `WebDB` *deductive database engine*, a generic database system with a highly customizable *html interface*. `WebDB` allows creating and maintaining Prolog databases stored in a variety of mediums by means of persistent predicates and using a WWW interface. They have also been used in real-world applications such as the Amos tool, aimed at facilitating the reuse of Open Source code through the use of an ontology-based search engine working on a large database of code information.

Full details are available in the full paper, where also experimental data and examples can be found [CGC+03].

Acknowledgments: This work has been partially supported by the EU IST Project IST-2001-34717, Amos and by MCYT project TIC 2002-0055, CUBICO. Thanks are due to I. Caballero, J. F. Morales, S. Genaim, and C. Taboch for their collaboration and feedback.

References

[CGC+03] J. Correas, J. M. Gomez, M. Carro, D. Cabeza, and M.V. Hermenegildo. A Generic Model for Persistence in CLP Systems. Technical Report CLIP3/2003.0, Technical University of Madrid, School of Computer Science, UPM, August 2003. `http://clip.dia.fi.upm.es/papers/persdb-tr.pdf`.